Detecting Refactorable Clones

Using Program Dependence Graph (PDG) and Program Slicing

Ammar Hamid
University of Amsterdam

Supervisor
Dr. Vadim Zaytsev

SATToSE 2014
L'Aquila, Italy

Research Scope

- A replication study of Komondoor-Horwitz 2001
- The goal is to validate result in original paper
- Interaction with the original writer
- Status: In progress

Research Context

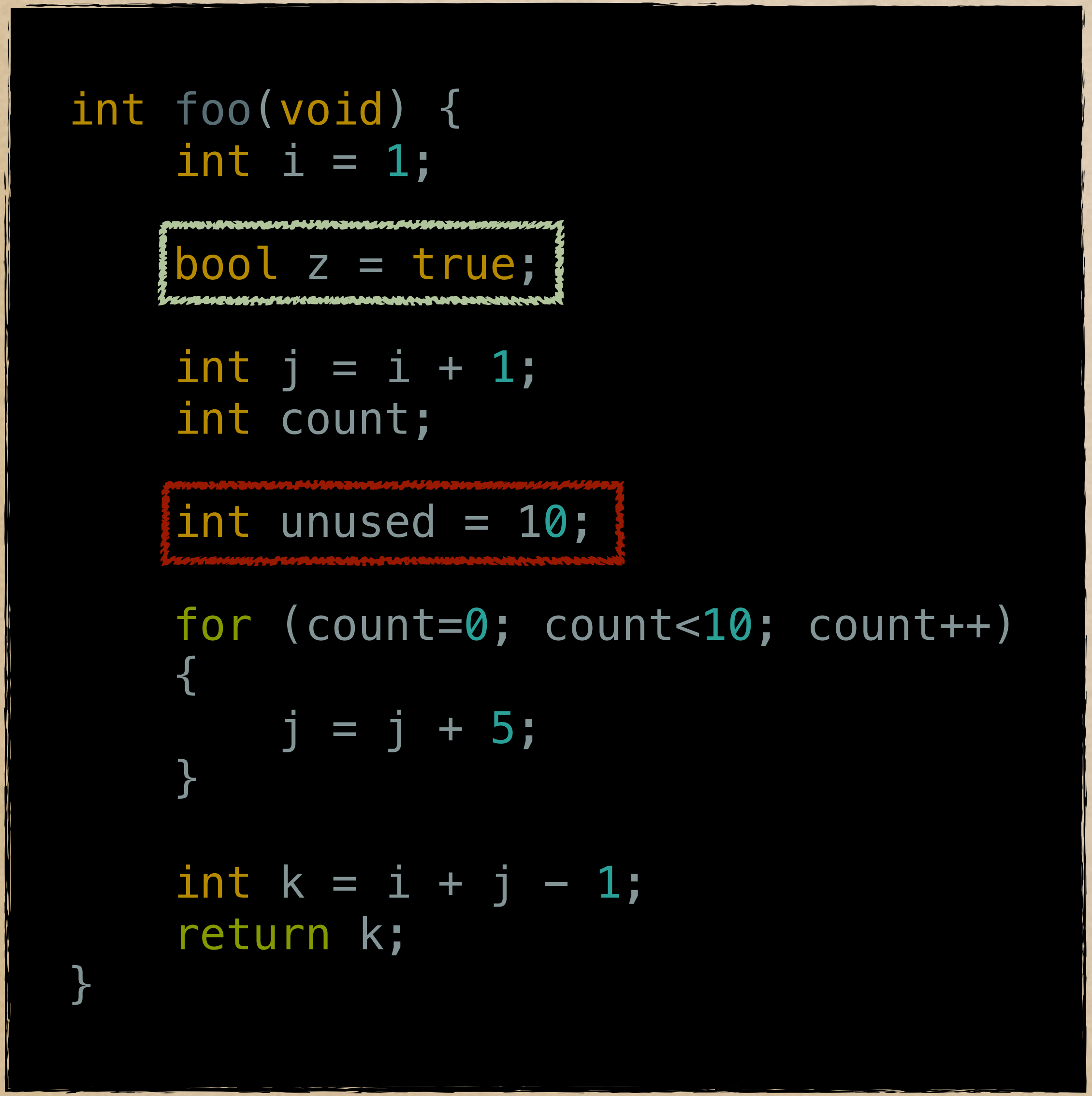
- Problem definition
- Semantic code clone detection - Type III

Algorithm

- Step 1: Find relevant procedures/methods
- Step 2: Find pair of vertices with equivalent syntactic structure
- Step 3: Find clones
- Step 4: Group clones

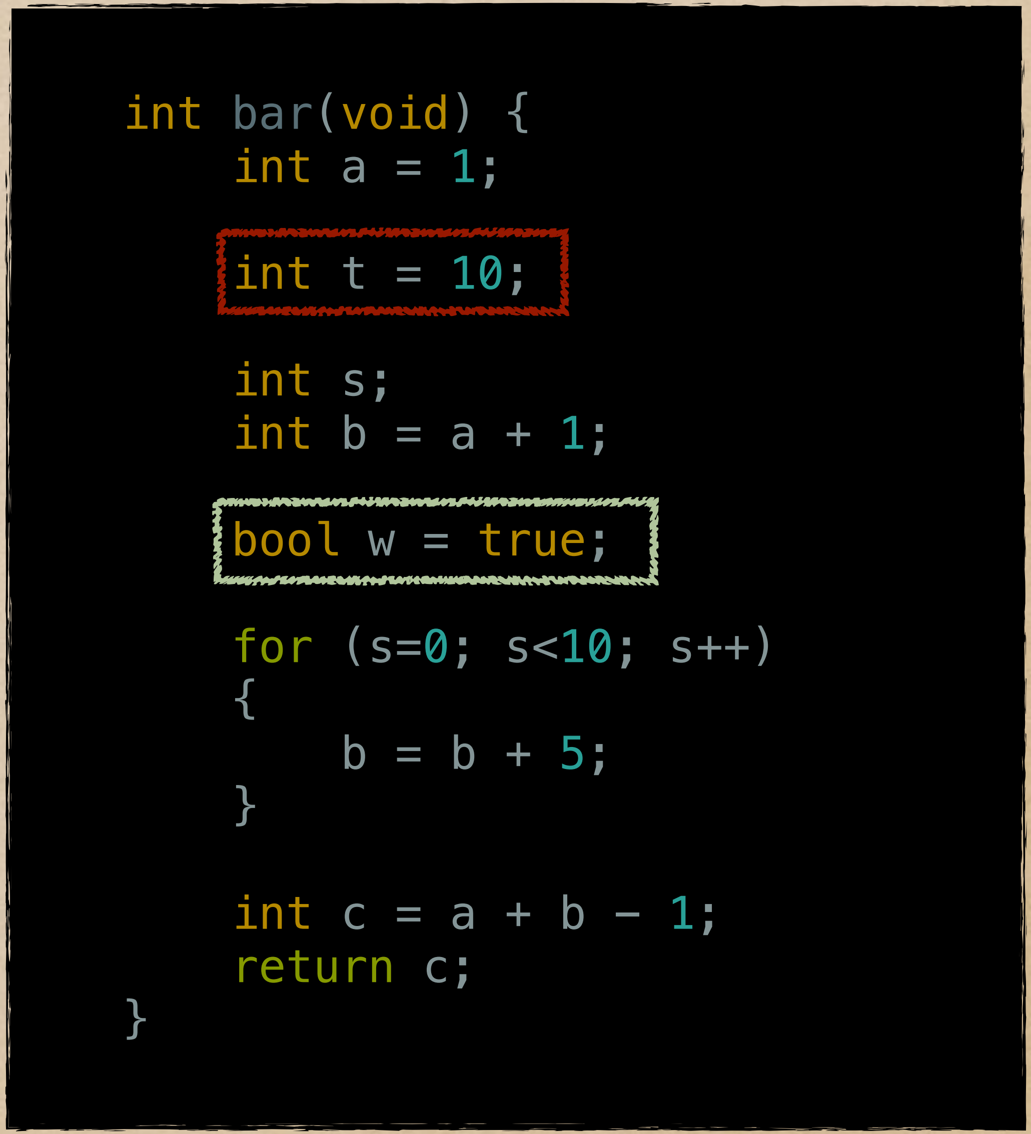
Example Of Clone Detection

Procedure A:



```
int foo(void) {
    int i = 1;

    bool z = true;

    int j = i + 1;
    int count;

    int unused = 10;

    for (count=0; count<10; count++)
    {
        j = j + 5;
    }

    int k = i + j - 1;
    return k;
}
```

Procedure B:



```
int bar(void) {
    int a = 1;

    int t = 10;

    int s;
    int b = a + 1;

    bool w = true;

    for (s=0; s<10; s++)
    {
        b = b + 5;
    }

    int c = a + b - 1;
    return c;
}
```

Implementation

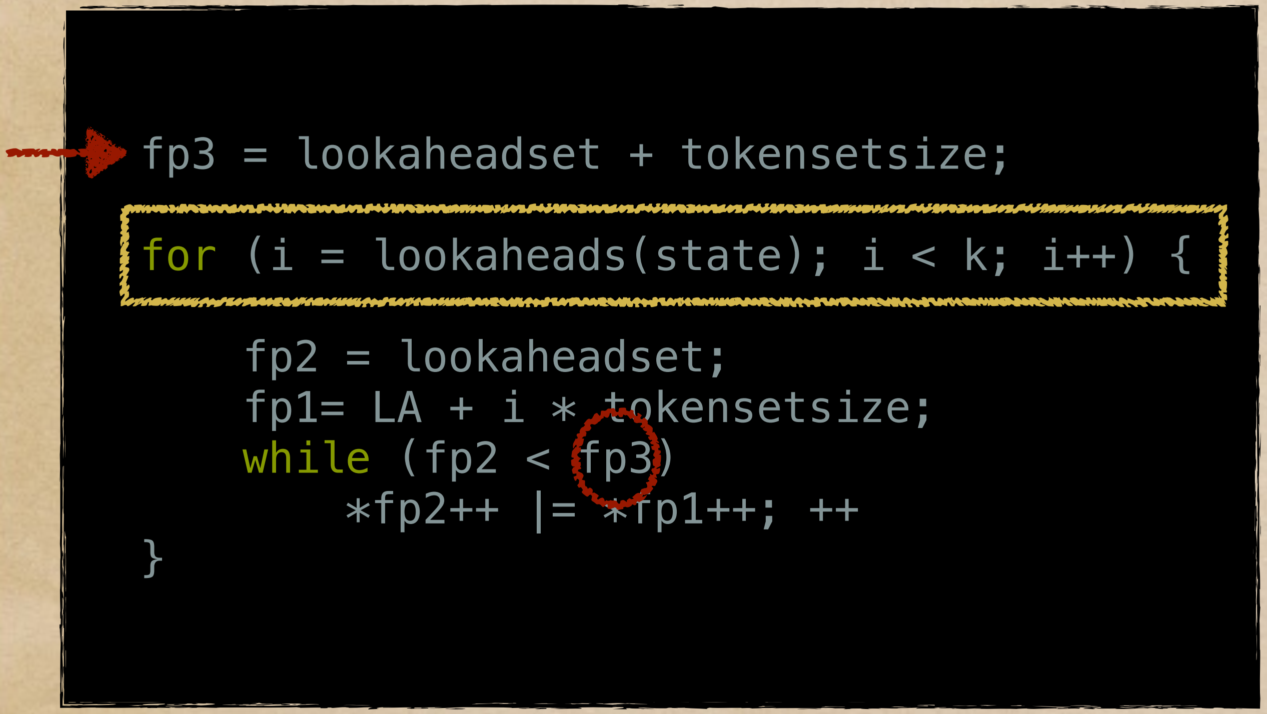
- Tools for generating PDG: CodeSurfer version 2.3
- 560 LOC Scheme
- Detect clones for C programs

Changes to the original study

- Only on reachable procedures
- No Forward-Slicing (see example in next slide)

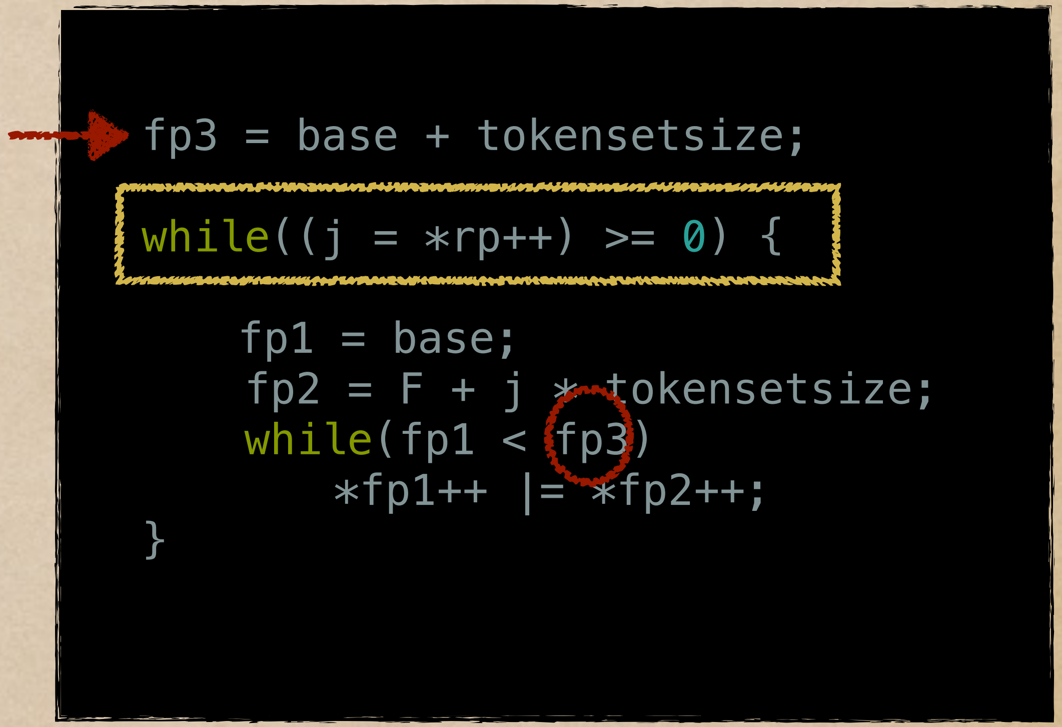
Why No Forward-Slicing:

Procedure A:



```
fp3 = lookaheadset + tokensetsize;
for (i = lookaheads(state); i < k; i++) {
    fp2 = lookaheadset;
    fp1= LA + i * tokensetsize;
    while (fp2 < fp3)
        *fp2++ |= *fp1++; ++
}
```

Procedure B:



```
fp3 = base + tokensetsize;
while((j = *rp++) >= 0) {
    fp1 = base;
    fp2 = F + j * tokensetsize;
    while(fp1 < fp3)
        *fp1++ |= *fp2++;
}
```

This is refactoring strategy - out of scope

Result so far

- TODO: run it on a real C program

Feedback

or

Questions